Modèle pour la redistribution de données complexes

Aurélien Esnard

RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants
Le Croisic, France, 5 au 8 avril 2005

Modèle pour la redistribution de données complexes

Aurélien Esnard

Projet ScAlApplix, INRIA Futurs et LaBRI UMR CNRS 5800,
351, cours de la Libération, F-33405 Talence - France
esnard@labri.fr

Résumé
Dans le cadre du couplage de codes, la redistribution efficace des données est un enjeu majeur pour obtenir de bonnes performances. Or la plupart des travaux dans ce domaine se limitent à l'étude d'objets simples comme des tableaux denses avec des distributions bloc-cycliques. Les applications multi-physiques modernes ou les environnements de pilotage (couplage simulation-visualisation) peuvent conduire à des schémas de redistribution plus irréguliers mettant en jeu des données complexes. Dans ce papier, nous nous écartons des modèles de redistributions classiques pour nous intéresser à une approche plus générique, basée sur la notion d'objet complexe. Grâce à une formulation ensembliste du problème, nous proposons un algorithme de redistribution parallèle pouvant s'appliquer à des objets spatiaux (tableaux denses ou creux, grilles structurées, particules) utilisant des distributions par bloc généralisées.

Mots-clés : distribution, redistribution, couplage de codes.

1. Introduction

Le problème de la redistribution des données a été beaucoup étudié dans le cadre de la programmation des architectures parallèles à mémoire distribuée, en particulier au titre des travaux menés autour de HPF [1] et de ScaLAPACK [2]. Sur ce type d'architecture, comme par exemple une grappe de PCs, le placement des données sur les processeurs est de première importance. En effet, c'est ce placement qui conditionne à la fois l'efficacité des algorithmes parallèles et la réduction du surcoût des communications. Le problème de la redistribution survient principalement dans deux contextes différents. Tout d'abord, il peut s'avérer utile au cours des étapes de calcul d'un programme parallèle de redistribuer les données pour optimiser la suite des calculs. Dans ce cas, la redistribution s'effectue << sur place >>, au sein du même code parallèle et chaque processeur envoie et reçoit des données. C'est par exemple le rôle de la directive *redistribute* dans le langage HPF. La redistribution des données est également primordiale dans le contexte du couplage de codes. Une application est alors vue comme un assemblage de plusieurs codes, le plus souvent parallèles, collaborant à la réalisation d'un travail commun (e.g. simulations multi-physiques). Ces codes sont typiquement distribués sur une grille de calcul, c'est-à-dire une grappe de calculateurs hétérogènes interconnectés par des réseaux plus ou moins performants. La communication entre les codes parallèles couplés nécessite de passer d'une distribution à une autre et donc de redistribuer les données (Fig. 1). On décompose généralement le problème de la redistribution des données en quatre étapes consécutives : la description des données distribuées, la génération des messages, l'ordonnancement des communications et le transfert effectif des messages.

Description des données distribuées. Le développement d'une solution standard au problème de la redistribution nécessite de définir un modèle de description également standard. Cette description doit spécifier d'une part la distribution des données entre les processeurs, et d'autre part l'agencement des données en mémoire sur chaque processeur.

Génération des messages. Le problème de la génération des messages consiste à déterminer, à partir des informations de distribution, les messages qui doivent être échangés entre chaque paire de processeurs. L'ensemble des messages ainsi calculés forme ce qu'on appelle la matrice de communication.

Ordonnancement. Une fois la matrice de communication calculée, il faut encore déterminer dans quel ordre effectuer les communications. En effet, les flux de communication parallèles, s'ils permettent d'agréger la bande-passante, peuvent entraîner des contentions réseaux qui dégradent les performances. Ainsi pour minimiser le temps de global de communication, il convient de découper les messages afin de les ordonnancer en étapes de communication.

Communication. Lors de l'étape de communication, les codes couplés échangent l'ensemble des messages générés en respectant l'ordonnancement calculé. Le schéma de communication global s'apparente à *all-to-all* personnalisé, transférant effectivement les données d'un code à l'autre.

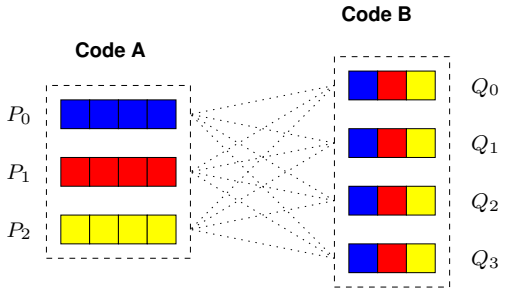

FIG. 1 – Le problème de la redistribution des données entre deux codes couplés.

Le problème de la génération des messages a principalement été étudié dans le cas des distributions bloc-cycliques de tableaux denses [3, 4, 5]. Ces distributions sont fréquemment utilisées dans le domaine du calcul scientifique car elles possèdent de bonnes propriétés d'équilibrage de charge. Dans ce contexte, il s'agit de passer d'une distribution bloc-cyclique à une autre distribution bloc-cyclique, ce qui conduit à un schéma de communication régulier. On trouve également beaucoup de résultats sur l'ordonnancement des communications pour ce type de distributions [6, 7]. Plus récemment, les auteurs de [8] ont proposé un algorithme d'ordonnancement général s'appliquant à tout type de distributions, uniquement basé sur la connaissance de la matrice de communication. De rares travaux abordent le problème de la redistribution dans un contexte aussi général. Dans PAWS [9] par exemple, il est possible de redistribuer des tableaux utilisant une décomposition rectilinéaire d'un domaine global. Dans CUMULVS [10], une simulation numérique parallèle peut être couplée à un ou plusieurs programmes séquentiels de visualisation. Les données considérées côté simulation sont uniquement des tableaux denses utilisant une distribution des données inspirée de HPF. Côté visualisation, les clients ont la possibilité de restreindre leur vue à un sous-domaine d'intérêt. CUMULVS propose également un modèle de données permettant de redistribuer des ensembles de particules (à coordonnées entières). Plus récemment, le projet CCA M×N [11] cherche à définir une interface de redistribution standard pour les composants logiciels, en intégrant plusieurs technologies dont celles de PAWS et de CUMULVS. Cependant ces travaux se limitent actuellement aux cas de données régulières.

Les travaux présentés dans ce papier sont largement motivés par le projet EPSN [12] qui cherche à définir un environnement pour le pilotage des simulations numériques, permettant de coupler une simulation parallèle avec un code de visualisation lui-même parallèle. A ce titre, nous nous intéressons

aux travaux de *Ahrens et al.* [13, 14] sur la visualisation parallèle des grands ensembles de données avec VTK [15]. Un tel couplage peut conduire à des redistributions de données plus irrégulières que celles étudiées par le passé, mettant en jeu des données complexes ne se limitant pas à des tableaux distribués en bloc-cyclique. En effet, les données de la simulation sont généralement distribuées pour optimiser les calculs numériques, tandis que du côté de la visualisation, la distribution tend plutôt à refléter la spatialité des données. A titre d'exemple, on peut considérer la simulation POP (Parallel Ocean Program) du projet CCSM (Community Climate System Model) [16]. Son modèle utilise une grille curvilinéaire de $3600 \times 2400 \times 40$ cellules pour représenter les océans du fond à la surface. Sur chaque niveau, la grille est découpée en blocs de taille 16×16, qui sont placés sur un ensemble de processeurs afin d'équilibrer la charge globale des calculs. De plus, les blocs entièrement recouvert de terre, sur les continents, ne sont pas pris en compte dans ce modèle (structure creuse). De manière générale, il s'avère important d'étendre les études précédentes, afin de pouvoir considérer des distributions de données plus générales, comme celles utilisées dans POP, mais également d'étendre les modèles afin de considérer des objets irréguliers comme par exemple des ensembles de particules ou des maillages non structurés.

Dans ce papier, nous nous intéressons au problème général de la redistribution des données complexes et plus précisément au calcul de la matrice de communication dans le cadre du couplage de codes. Nous n'abordons pas ici le problème de l'ordonnancement des communications, pour lequel nous souhaitons reposer sur des résultats existants [8]. Dans la section suivante, nous proposons une formulation ensembliste du problème de la redistribution, très générale, servant de fil directeur à cette étude. Puis, nous introduisons les notions d'objet complexe et d'objet spatial distribué par bloc (section 3), pour lesquels nous décrivons un algorithme de redistribution adapté (section 4). Pour terminer, nous validons notre approche en présentant quelques résultats expérimentaux obtenus avec les bibliothèques RedSYM et RedCORBA.

2. Formulation ensembliste du problème de la redistribution

Soit P un ensemble de processeurs de taille M. Soit E un ensemble d'éléments totalement ordonnés. On associe à chaque élément de E un indice global k et on note $E[k]$ le k-ème élément de cet ensemble. Par définition, *une décomposition* de E sur P est un ensemble de M sous-ensembles de E, noté $E/P = \{A_0, A_1, \ldots, A_{M-1}\}$, et tel que les éléments du sous-ensemble A_i sont associés au processeur P_i. Dans la littérature, les décompositions considérées forment le plus souvent une partition de E. Dans notre étude, les ensembles A_i sont quelconques et peuvent tout à fait être vides, se recouper ou ne pas couvrir tout l'ensemble E. Par définition, *une distribution* de E sur P est une fonction $\alpha : k \longrightarrow (i, l)$ qui à l'indice global k d'un élément de E associe un processeur de rang i dans P et un indice local l dans A_i. On dit alors que E est distribué sur P selon α et on note $A_i[l]$ l'élément d'indice local l placé sur le processeur P_i. Sur chaque A_i, les indices locaux forment un ordre total. Ainsi, on a $E[k] = A_i[l]$ si et seulement si $\alpha(k) = (i, l)$. Dans le cas particulier d'une distribution bloc-cyclique 1D, la fonction de distribution s'écrit : $k \longrightarrow (\lfloor k/s \rfloor \bmod M,\ b.s + k \bmod s)$ avec s la taille du bloc, $b = \frac{\lfloor k/s \rfloor}{M}$ l'indice local du bloc relatif à k et M le nombre total de processeurs [6].

Soient E un ensemble d'éléments, P et Q deux ensembles de processeurs respectivement de taille M et N. Considérons la distribution α de E sur P et β de E sur Q, avec $E/P = \{A_0, A_1, \ldots, A_{M-1}\}$ et $E/Q = \{B_0, B_1, \ldots, B_{N-1}\}$. Dans cette étude, nous considérons uniquement le cas où deux distributions *a priori* différentes sont appliquées à même ensemble d'éléments. Ce prédicat essentiel permet de garantir que le problème de la redistribution à une solution unique, puisqu'il est alors possible de mettre en correspondance les éléments un à un entre les sous-ensembles A_i et B_j. Pour résoudre le problème de la redistribution, il faut calculer les messages échangés entre P et Q. On note $m_{i,j}$ *le message* échangé entre P_i et Q_j. Ce message contient l'ensemble des éléments résultant de *l'intersection* entre A_i et B_j (Fig. 2). Si E/P et E/Q sont des partitions de E, alors l'ensemble des messages $\{m_{i,j}\}$ (non vide) défi-

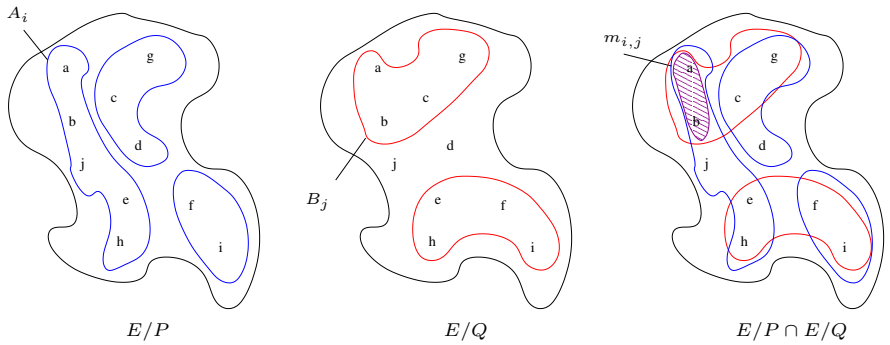

FIG. 2 – Décomposition de l'ensemble d'éléments E sur P et Q. Calcul du message $m_{i,j}$ par intersection des ensembles A_i et B_j.

nit une nouvelle partition de E, plus fine que les partitions E/P et E/Q. Comme l'illustre la figure 3, un élément e appartient au message $m_{i,j}$, si et seulement si il existe un entier k tel que $\alpha(k) = (i, l)$ et $\beta(k) = (j, l')$, c'est-à-dire tel que $e = E[k] = A_i[l] = B_j[l']$. On note $\gamma_{i,j} : l \longrightarrow l'$ l'application associant l'index local l d'un élément dans A_i à son index local l' dans B_j. Par extension, le schéma de redistribution est complètement défini par l'application $\gamma : (i, l) \longrightarrow (j, l')$ vérifiant $\gamma_{i,j}(l) = l'$. L'ensemble des messages échangés entre chaque paire de processeurs (P_i, Q_j) forme *la matrice de communication de P vers Q*, notée $\mathbb{M} = (m_{i,j})$ de taille $M \times N$. Une ligne i de la matrice désigne l'ensemble des messages que le processeur P_i doit envoyer vers Q. A l'inverse, une colonne j de la matrice correspond à l'ensemble des messages reçus par le processeur Q_j.

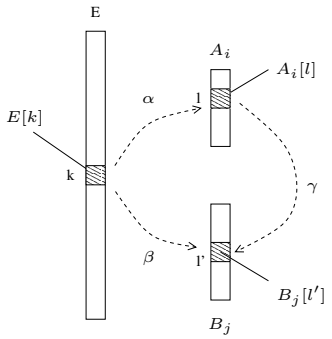

FIG. 3 – Redistribution de P vers Q selon γ.

3. Les objets complexes

Le modèle de description des données que nous proposons est basé sur la notion *d'objet complexe*. Ce modèle a pour objectif de prendre en compte une grande variété de structures de données présentes dans les codes de calcul scientifique, tout en permettant une génération efficace des messages pour la redistribution. Par définition, un objet complexe $\mathcal{O}$ est une application d'un ensemble d'éléments E vers un ensemble de valeurs $\mathcal{V}$, généralement constitué de plusieurs *séries* de données numériques (entières ou réelles). Les séries sont un moyen simple d'associer plusieurs valeurs aux éléments d'un objet (e.g. pression-température ; position-vitesse-force, *etc.*). De plus, l'utilisation des séries nous permet de factoriser le calcul de la redistribution. La distribution des éléments, telle que nous l'avons définie dans la

section précédente, s'étend naturellement aux objets complexes, induisant une distribution des données associées aux éléments. Ainsi, chaque processeur P_i possède dans sa mémoire locale un ensemble de valeurs correspondant aux éléments A_i. Sur chaque processeur P_i, nous considérons un niveau de subdivision supplémentaires, appelé *région*. Par définition, une région est un sous-ensemble des éléments de A_i, tels que l'ensemble des régions associées à processeur P_i forment une partition de A_i. On note $R_{i,j}$ la j-ème région du processeur P_i. Les régions sont utiles pour regrouper des éléments connectés logiquement comme par exemple les composantes connexes d'un maillage ou les blocs d'éléments dans un tableau multidimensionnels. Par ailleurs, elles permettent de considérer le problème de la redistribution avec une granularité moins fine que l'élément.

Sur la base de ce modèle, nous avons proposé la définition de plusieurs classes d'objets : les champs (tableaux), les grilles structurées, les ensembles de particules, les maillages non structurés, *etc*. Dans cet article, nous nous intéressons uniquement à la famille des *objets spatiaux distribués par bloc* pour lesquels nous proposons un algorithme de redistribution unifié dans la section 4.

3.1. Objets spatiaux distribués par bloc

Les objets spatiaux distribués par bloc sont des objets très fréquents dans les codes de simulation numérique parallèle. Ils permettent principalement de représenter des champs (tableaux), des grilles structurées et des ensembles de particules (Fig. 4).

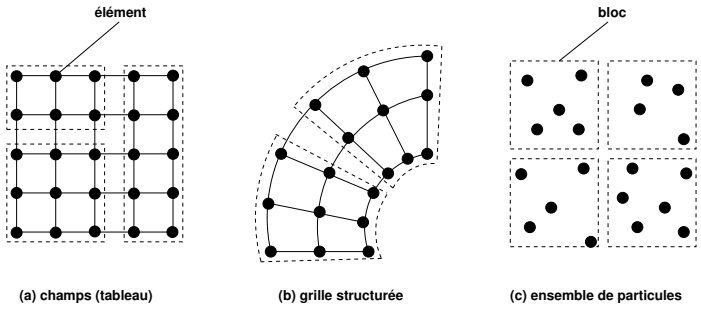

FIG. 4 – Les objets spatiaux distribués par bloc.

Par définition, un objet O est dit spatial dans $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$) si il existe *une fonction coordonnée* φ de E vers $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$), telle que pour tout élément $e \in E$, $\varphi(e) = (x, y, ...)$ désigne la coordonnée de e. Un *bloc* B est un parallélépipède rectangle de l'espace des coordonnées, décrit par deux points extrêmes (A, B) tels que $A = (x_a, y_a, \ldots)$ et $B = (x_b, y_b, \ldots)$. Considérons un ensemble de processeurs P. Une distribution par bloc sur P est une fonction δ_P qui à chaque processeur d'indice i associe un ensemble de blocs $S_i = \{B_{i,0}, B_{i,1}, \ldots\} = \delta_P(i)$. Par définition, un objet spatial O est *distribué par bloc* selon δ_P si, pour tout élément e appartenant à la région $R_{i,j}$, on vérifie $\varphi(e) \in B_{i,j}$. Ainsi les blocs jouent le rôle de conteneurs pour les éléments, et à chaque bloc $B_{i,j}$, il est possible d'associer une et une seule région $R_{i,j}$ sur P_i.

Sur la base de ce modèle, nous avons proposé la définition de plusieurs classes d'objets spatiaux distribués par bloc, dont nous allons maintenant présenter les caractéristiques. Tout d'abord, *les champs* sont des tableaux multidimensionnels possédant un système de coordonnées naturel $(i, j, ...)$ dans $\mathbb{Z}^n$. En général, l'ensemble des blocs $B_{i,j}$ résultent d'une décomposition en sous-blocs d'un domaine initial D, chaque S_i formant ce qu'on appelle un sous-domaine de D. Notons qu'il n'est fait aucune hypothèse dans ce modèle sur l'arrangement des blocs. Ainsi chaque processeur peut posséder un ensemble de blocs tout à fait quelconque ou ne pas en posséder du tout. Contrairement aux travaux précédents, nous

ne nous limitons pas à des distributions rectilinéaires ou simplement en bloc-cyclique. Nous étendons le modèle de distribution à une représentation par bloc plus générique, ce qui rend notre approche très flexible, même si la description des distributions implique un surcoût mémoire (stockage explicite des coordonnées des blocs). Les grilles structurées sont des objets à topologie régulière, très proches des champs, formées de quadrilatères en 2D et d'hexaèdres en 3D. Tout comme les champs, ces objets possèdent un système de coordonnées naturel $(i, j, ...)$ dans $\mathbb{Z}^n$, que nous utilisons pour décrire les distributions par blocs. De plus, une coordonnée physique dans $\mathbb{R}^n$ est associée à chaque nœud de la grille. En pratique, les champs sont suffisants pour représenter des grilles structurées comme celles utilisées dans la simulation POP, car il suffit d'utiliser une série de données particulière pour stocker les coordonnées physiques de la grille (nécessaire par exemple à la visualisation du modèle dans EPSN). Les ensembles de particules que nous considérons sont des objets spatiaux dans $\mathbb{R}^n$, utilisant la position physique des particules dans l'espace comme fonction coordonnée. Ce type d'objet est très fréquent dans les codes de dynamique moléculaire, comme par exemple NAMD [17]. Dans cette simulation, les particules sont distribuées selon un pavage de l'espace $\mathbb{R}^3$. A chaque particule est associée en plus de la position, une vitesse et une force. Dans le cas des champs ou des grilles, le nombre d'éléments est contraint par la taille des blocs. En revanche dans le cas des particules, le nombre d'éléments contenus dans les blocs fluctue au cours de la simulation.

3.2. Modèle de stockage

Considérons un objet $\mathcal{O}$ constitués de s séries de données. Pour chaque série, il est nécessaire de décrire le stockage des données en mémoire, afin de pouvoir construire physiquement les messages calculés. Le modèle que nous utilisons est basé sur une approche classique en *stride*. Pour accéder physiquement à la valeur des éléments en mémoire, il faut se donner s fonctions de *mapping*, une par série. Par définition, une fonction de *mapping* $v : l \longrightarrow a$ associe à chaque élément d'indice local l sur P_i une adresse physique a dans la mémoire du processeur. Considérons un espace de stockage L à p dimensions, de taille $L_0 \times L_1 \times ... \times L_{p-1}$. A chaque dimension est associée un espacement ou *stride* (noté T_i, en octets) qui permet de passer à l'élément suivant selon la i-ème direction de l'espace (Fig. 5). La fonction de *mapping* résultante s'écrit : $a = A + \mathit{off} + c_0.T_0 + c_1.T_1 + ... + c_{p-1}.T_{p-1}$, avec A l'adresse de base, *off* l'offset initial et (c_i) la coordonnée de stockage obtenue par simple division euclidienne de l dans l'espace de stockage (L_i). Ce modèle ne permet pas d'accéder à une séquence d'adresses mémoire quelconque, toutefois il est bien adapté à notre problématique, car il permet de prendre en compte une grande variété de structures de données (e.g. données continues, données filtrées, *ghost*, *etc.*), tout en offrant un accès aléatoire à la mémoire très efficace (en $\mathcal{O}(p)$ avec $p \simeq n$ dans le cas général et $p = 1$ si les données sont continues).

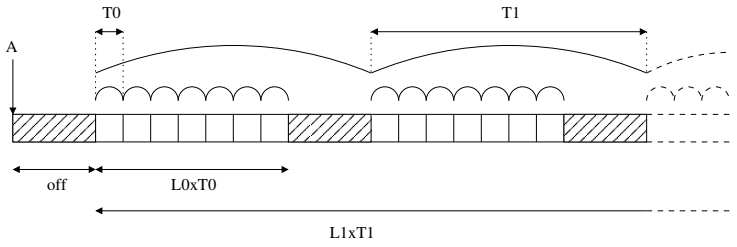

FIG. 5 – Modèle de stockage mémoire en *stride*.

4. Algorithme de redistribution spatial par bloc

Dans cette section, nous présentons un algorithme parallèle pour le calcul de la matrice de communication $\mathbb{M} = (m_{i,j})$. Il ne s'agit donc pas ici de calculer un ordonnancement des communications ni même

d'effectuer ces communications. Cet algorithme s'applique à tout type d'objets spatiaux distribués par bloc (e.g. champs, grille, particules) et repose sur une *opération abstraite* d'extraction des données dépendant de la classe de l'objet considéré.

Soient deux ensembles de processeurs P et Q, respectivement de taille M et N. Soit E un ensemble d'éléments et $\mathcal{O}$ un objet spatial distribué par bloc sur P et Q, selon les distributions δ_P et δ_Q. Chaque processeur P_i possède uniquement les informations locales de l'objet, notées $\mathcal{O}/P_i$, c'est-à-dire la description de l'ensemble des blocs $\delta_P(i) = \{B_{i,r}\}$ et les fonctions *mapping* permettant d'accéder directement aux données en mémoire. Pour chaque processeur P_i, l'algorithme que nous proposons calcule l'ensemble des messages à échanger avec tous les Q_j, ce qui revient uniquement à calculer sur chaque P_i une ligne de la matrice de communication $m_{i,*}$. Notons que dans notre approche, il n'est en aucun cas nécessaire de centraliser la matrice de communication, qui reste donc distribuée. Toutefois, le calcul de la redistribution nécessite pour chaque P_i de connaître la distribution distante, c'est-à-dire l'ensemble des blocs $\delta_Q(j) = \{B_{j,l}\}$ pour tous les Q_j. L'acquisition de cette information implique une étape de communication préliminaire où tous les processeurs Q_j communiquent avec tous les processeurs P_i. La génération des messages pour les Q_j s'effectue de manière tout à fait symétrique aux P_i.

Algorithm 1

```
1   for each processor i ∈ P do in parallel
2       for each remote processor j ∈ Q do
3           for each block B_{i,r} ∈ δ_P(i) do
4               for each block B_{j,l} ∈ δ_Q(j) do
5                   Π_{i,j} ← Π_{i,j} ∪ (B_{i,r} ∩ B_{j,l})
6               end for
7           end for
8           reorder blocks in Π_{i,j}
9           for each block B_k ∈ Π_{i,j} do
10              m_{i,j} ← m_{i,j} ∪ (extract B_k from O/P_i)
11          end for
12      end for
13  end for
```

FIG. 6 – Algorithme de redistribution spatial par bloc.

Cet algorithme (Fig. 6) se décompose en trois étapes : (a) le calcul des blocs de redistribution par intersection géométrique (ligne 3 à 7), (b) le tri des blocs de redistribution dans un ordre canonique (ligne 8), et (c) la génération des messages par extraction des buffers (ligne 9 à 11). Le principe d'intersection formulé dans la section 2 s'applique naturellement aux objets spatiaux distribués par bloc. Comme le montre la figure 7, l'intersection géométrique des blocs locaux et distants (dans $\mathbb{Z}^n$ ou $\mathbb{R}^n$) permet de calculer simplement l'ensemble des *blocs de redistribution*, noté $\Pi_{i,j}$, qu'il faut échanger entre chaque couple de processeurs (P_i, Q_j). Le message $m_{i,j}$ est alors formé des données contenues par l'ensemble des blocs de redistribution $\Pi_{i,j}$ (Fig. 8). Afin de sérialiser les données constituant ce message, il est nécessaire d'ordonner canoniquement les blocs de redistribution dans $\Pi_{i,j}$ (*reorder*). Il suffit alors d'extraire les éléments contenus dans chaque bloc de redistribution (*extract*) et de les concaténer pour former un message. Sans l'utilisation d'un ordre canonique commun à P_i et Q_j, la sérialisation des données pourrait s'effectuer dans une ordre différent entre les processeurs communiquants. L'ordre sur les blocs que nous utilisons est simplement basé sur la comparaison de la position des blocs dans l'espace des coordonnées.

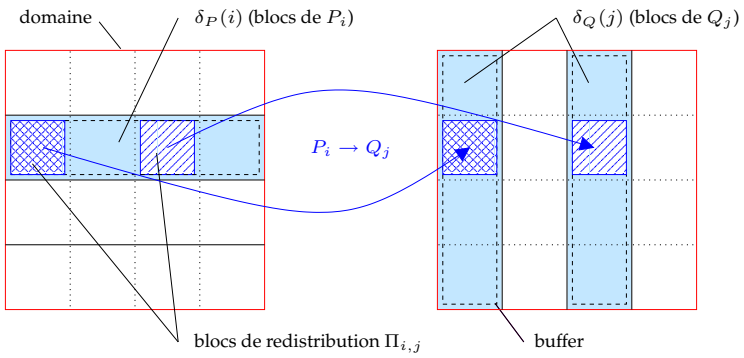

FIG. 7 – Calcul des blocs de redistribution $\Pi_{i,j}$ par intersection des blocs de $\delta_P(i)$ et $\delta_Q(j)$.

Dans cet algorithme, l'opération prépondérante est l'extraction des données générant le message, dont la complexité dépend de la nature des objets manipulés. Le nombre d'extractions à effectuer est égal à $\|\Pi_{i,j}\|$ dont la valeur est bornée par le produit du nombre de blocs locaux et du nombre total de blocs distants, c'est-à-dire par $\|\delta_P(i)\| \times \prod_j \|\delta_Q(j)\|$. Dans le cas des champs ou des grilles, l'extraction est immédiate, même si elle peut entraîner un découpage important des données en mémoire (Fig. 8-a). En revanche pour les particules, il est nécessaire de tester les coordonnées une à une, lorsque les distributions sont << pathologiques >>, c'est-à-dire lorsque l'intersection entre un bloc local et distant est partielle (ni vide, ni totale), comme l'illustre la figure 8-b. Notons que lorsque les distributions considérées sont à grain suffisamment fin (cas d'un pavage de l'espace), les cas pathologiques disparaissent et l'extraction devient triviale, dans le sens où il suffit de transférer des blocs entiers sans avoir à trier les particules à l'intérieur.

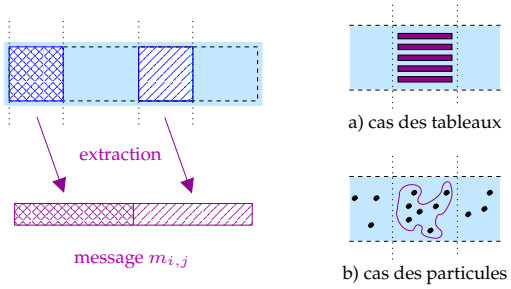

FIG. 8 – Génération du message $m_{i,j}$ par extraction des buffers correspondants à $\Pi_{i,j}$.

5. Validation

5.1. Les bibliothèques RedSYM & RedCORBA

Afin de valider notre approche, nous avons développé un environnement logiciel pour le couplage de codes parallèles et plus précisément pour la redistribution des objets complexes. Cet environnement se

compose de deux bibliothèques C++, appelées RedSYM et RedCORBA.

La bibliothèque RedSYM implante le modèle de description des données et les algorithmes présentés dans ce papier. Plus précisément, RedSYM calcule (en parallèle) la matrice de communication à partir de la description des objets complexes. Cette bibliothèque est dite *symbolique* dans le sens où son interface est clairement indépendante d'une couche de communication. Dans RedSYM, les messages générés sont uniquement stockés de manière symbolique sous la forme d'un *masque*, c'est-à-dire d'une séquence d'intervalles d'indices locaux désignant les éléments formant un message. L'association d'un masque à un objet complexe permet de reconstruire efficacement la séquence des adresses mémoire constituant le message physique.

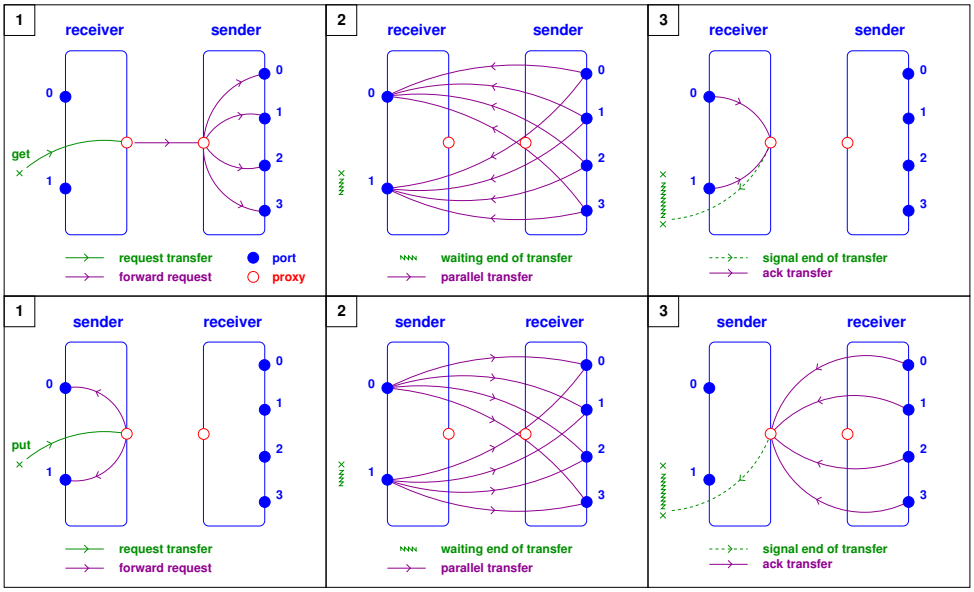

FIG. 9 – Requêtes *get* et *put* dans RedCORBA.

La bibliothèque RedCORBA implante une couche de communication au dessus de RedSYM, basée sur la technologie CORBA. Elle permet de coupler dynamiquement plusieurs codes parallèles (e.g. MPI, PVM, *etc.*) sur un réseau hétérogène. Elle prend en charge l'échange des descriptions entre les codes couplés, et le transfert parallèle des données d'un code (*sender*) à l'autre (*receiver*). RedCORBA utilise une approche composant, comparable à celle mise en œuvre dans PAWS [9], basée sur la notion de *ports* de communication et de *data channel*. Par ailleurs, un *proxy* permet de coordonner les transferts via un système de requêtes *get/put*, représenté sur la figure 9. Ce système implique une gestion centralisée des requêtes et des acquittements, alors que transfert s'effectue toujours en point-à-point entre les ports. Afin d'optimiser les performances lors des transferts successifs, nous stockons la matrice de communication symbolique calculée par RedSYM. Les messages CORBA générés par RedCORBA sont également stockés sous la forme d'une séquence d'adresses mémoire. Le transfert des messages repose sur une stratégie zéro-copie n'imposant pas de copie des données à l'envoi. A la réception, une copie est toujours nécessaire dans CORBA. Par ailleurs, lorsque les messages sont trop découpés en mémoire, il est possible d'activer la gestion automatique d'un cache dans RedCORBA, qui sera remis à jour avant chaque transfert. Nous soulignons le fait que la matrice de communication est calculée une seule fois, lors de l'étape de connexion des codes couplés et qu'elle est ensuite conservée durant toutes les étapes

de communication. Cette matrice devra être recalculée uniquement dans le cas où la distribution des données est modifiée.

5.2. Résultats préliminaires

Présentons maintenant quelques résultats expérimentaux sur la redistribution des champs et des particules obtenus avec RedCORBA. Les prises de performance ont été réalisées sur un cluster de 16 bixeons interconnectés par un réseau Giga-Ethernet (bande-passante théorique à 128 Mo/s). Nous utilisons OmniORB4 [18], un ORB réputé pour ses bonnes performances : débit maximal mesuré à 108 Mo/s (latence $80\ \mu s$) en point à point sur ce réseau contre 112 Mo/s pour LAM/MPI. Nous évaluons le transfert de données réelles (*double*) entre deux codes parallèles (LAM/MPI) couplés avec RedCORBA, l'un jouant le rôle d'émetteur (M processeurs P_i) et l'autre de récepteur (N processeurs Q_j). Plus précisément, nous mesurons le débit cumulé moyen (volume total / temps total en Mo/s) obtenue avec un requête *put* pour différents nombres de processeurs (2×2, 4×4, 8×8, 16×16 et 16×10) et différentes tailles de données grossièrement compris entre 128 o et 200 Mo (i.e. tableau 5000×5000, 10 millions de particules 2D). Dans l'expérience sur les champs, nous considérons d'un côté une distribution bloc-ligne 1D d'un tableau dense 2D et de l'autre une distribution bloc-colonne 1D. Nous avons choisi ces deux distributions car elles impliquent un schéma de communication total entre ces deux codes et un découpage des données en mémoire relativement important. Pour l'expérience sur les particules, nous utilisons les mêmes motifs de distribution appliqués à un domaine de $\mathbb{R}^2$ (de taille fixe), mais nous raffinons encore ce découpage. D'un côté, chaque processeur P_i possède une colonne de N blocs et de l'autre côté, chaque processeur Q_j possède une ligne de M blocs, de telle façon que l'intersection des distributions soit non pathologique si $M = N$ (i.e. pas de tri des particules). Par ailleurs, nous considérons que les particules se déplacent, ce qui impose de mettre à jour le buffer de communication avant chaque transfert. Les résultats obtenus pour les champs (Fig. 10) et les particules (Fig. 11) sont tout à fait satisfaisants, car ils montrent nettement l'agrégation de la bande-passante obtenue avec RedCORBA. Notons que le débit cumulé mesuré est divisé par deux par rapport au nombre de processeurs utilisés car nous utilisons des bi-processeurs, ne possédant qu'une seule interface réseau. Ces résultats sont d'autant plus satisfaisants que ces mesures prennent en compte le temps de gestion des requêtes et des acquittements de RedCORBA (temps d'aller-retour à vide d'un requête évaluée à $160\ \mu s$ en 1×1).

6. Conclusion

Dans ce papier, nous nous sommes écartés des modèles de redistribution classiques en bloc-cyclique pour nous intéresser à une approche plus générique basée sur la notion d'objet complexe. Grâce à une formulation ensembliste du problème, nous avons proposé un algorithme de redistribution applicable aux objets spatiaux distribués par bloc tels que les champs, les grilles structurées, et les ensembles de particules. L'approche spatiale étant mal adaptée pour la redistribution des maillages non structurés, nous évaluons actuellement une nouvelle approche basée sur des heuristiques de partitionnement et de placement, pouvant également s'appliquer à des ensembles de particules, et possédant de bonnes propriétés pour le couplage simulation-visualisation. Les algorithmes présentés dans ce papier sont déjà utilisés dans l'environnement de pilotage EPSN [12], grâce aux bibliothèques RedSYM et RedCORBA. Par ailleurs, nous collaborons activement au sein de l'ARC RedGRID pour intégrer ces travaux dans la plate-forme PaCO++ [19].

Bibliographie

1. C. Koebel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel. *The High Performance Fortran Handbook*. MIT Press, Cambridge, 1994.
2. L.S. Blackford, J. Choi, A. Cleary, E. D'Azevado, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling,

G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. ScaLAPACK Users'Guide. Technical report, SIAM, 1997.
3. L. Prylli and B. Tourancheau. Efficient Block Cyclic Data Redistribution. In *Euro-Par*, volume 1, pages 155–164. LNCS Springer Verlag, August 1996.
4. R. Thakur, A. Choudhary, and J. Ramanujam. Efficient algorithms for array redistribution :. *IEEE Transactions on Parallel and Distributed Systems*, 7(6) :587–594, 1996.
5. D. W. Walker and S. W. Otto. Redistribution of block-cyclic data distributions using MPI. *Concurrency : Practice and Experience*, 8(9) :707–728, 1996.
6. F. Desprez, J.J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. *IEEE Transaction on Parallel and Distributed Systems*, 9(2) :192–205, 1998.
7. F. Desprez, S. Domas, J.J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. More on Scheduling Block-Cyclic Array Redistribution. In *Proceedings of 4th Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers (LCR98)*, volume 1151, pages 275–287. LNCS Springer-Verlag, 1998.
8. E. Jeannot and F. Wagner. Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone. *IPDPS*, 2004.
9. Peter H. Beckman, Patricia K. Fasel, William F. Humphrey, Susan M. Mniszewsk. Efficient coupling of parallel applications using PAWS. *The Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC)*, page 215, 1998.
10. J. A. Kohl and P. M. Papadopoulos. CUMULVS : Providing fault-tolerance, visualization, and steering of parallel applications. *Int. J. of Supercomputer Applications and High Performance Computing*, pages 224–235, 1997.
11. CCA MxN. http ://www.csm.ornl.gov/cca/mxn.
12. A. Esnard, M. Dussere, and O. Coulaud. A Time-coherent Model for the Steering of Parallel Simulations. In *Euro-Par 2004 Parallel Processing*, pages 90–97. Springer LNCS, 2004.
13. J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka. A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets. Technical Report LAUR-001630, Los Alamos National Laboratory, 2000.
14. C. C. Law, A. Henderson, and J. Ahrens. An application architecture for large data visualization : A case study.
15. W. Schroeder, K. Martin, and B. Lorensen. *The Visualisation ToolKit*. Kitware, 2002.
16. Los Alamos National Laboratory. POP : Parallel Ocean Program. climate.lanl.gov/Models/POP.
17. Theoretical and Computational Biophysics Group. NAMD : Scalable Molecular Dynamics. http ://www.ks.uiuc.edu/Research/namd.
18. OmniORB. http ://omniorb.sourceforge.net.
19. C. Pérez, T. Priol, and A. Ribes. PaCO++ : A parallel object model for high performance distributed systems. In *Distributed Object and Component-based Software Systems Minitrack in the Software Technology Track of the 37th Hawaii International Conference on System Sciences (HICSS-37)*, Big Island, Hawaii, USA, January 2004. IEEE Computer Society Press.

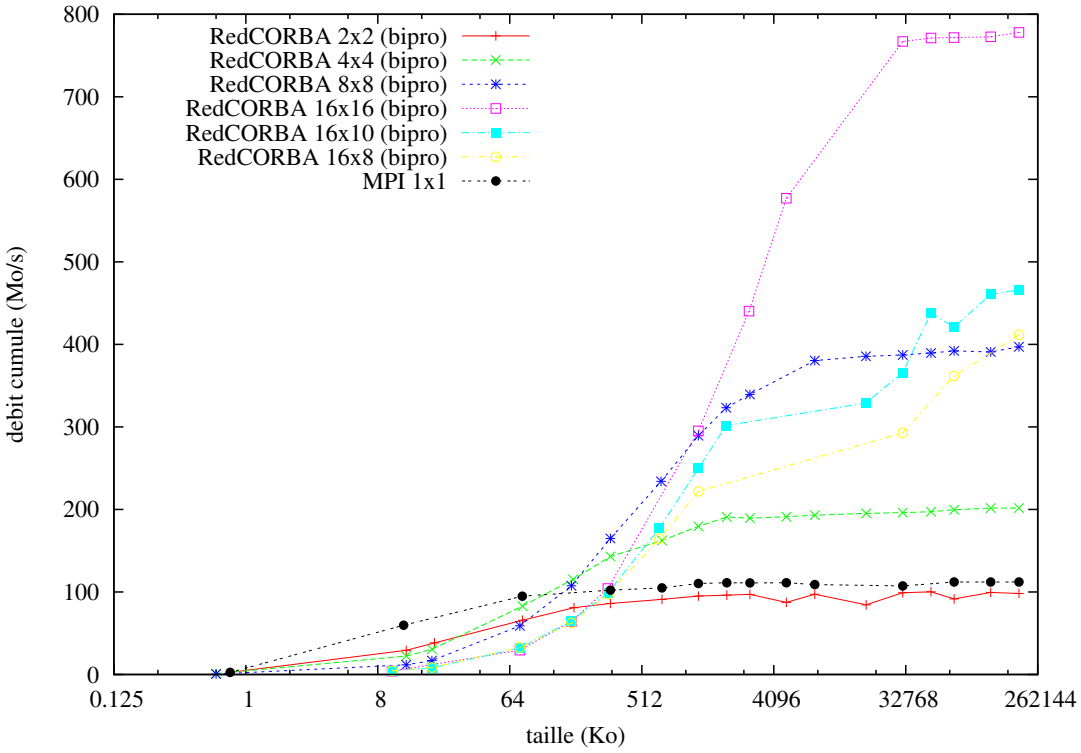

FIG. 10 – Redistribution avec RedCORBA de tableaux denses 2D.

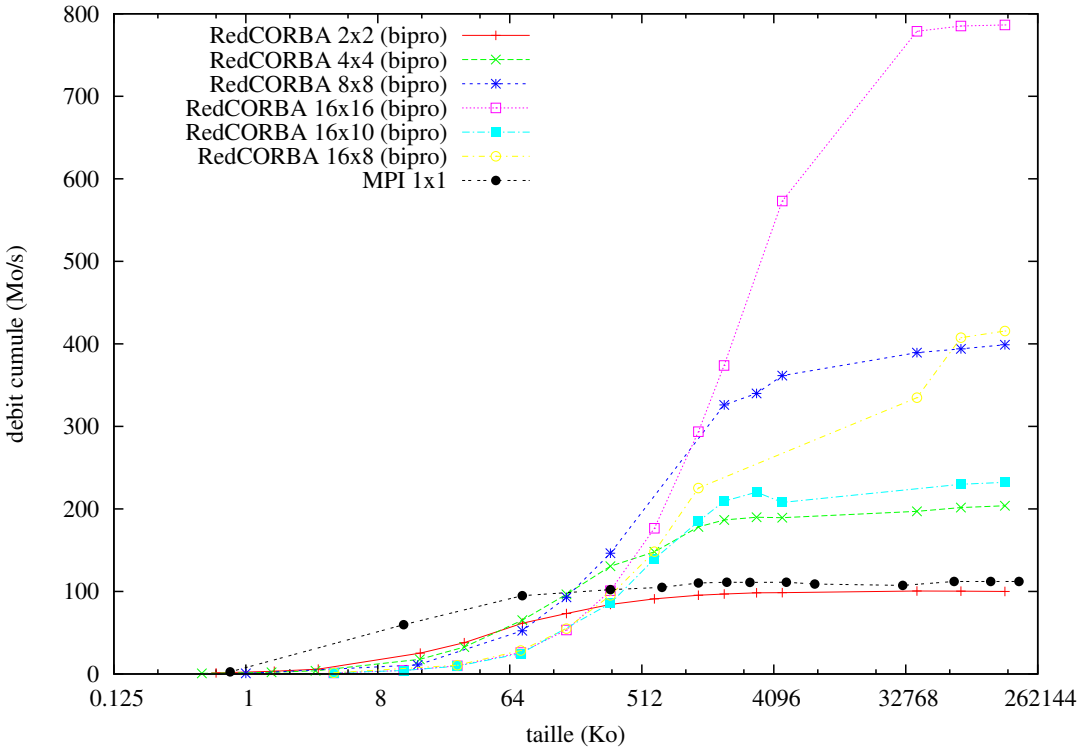

FIG. 11 – Redistribution avec RedCORBA de particules dans $\mathbb{R}^2$.